NASA TM-85395

SOFTWARE ENGINEERING LABORATORY SERIES

SEL-82-005

GLOSSARY OF SOFTWARE ENGINEERING LABORATORY TERMS

DECEMBER 1982

NASA

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

NASA TM-85395

SOFTWARE ENGINEERING LABORATORY SERIES

SEL-82-005

GLOSSARY OF SOFTWARE ENGINEERING LABORATORY TERMS

DECEMBER 1982

NASA

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

N83-32354 #

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC (Systems Development and Analysis Branch)
The University of Maryland (Computer Sciences Department)
Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-82/6214.

The contributors to this document include

Michael Rohleder	(Computer Sciences Corporation)
Frank McGarry	(Goddard Space Flight Center)
Jerry Page	(Computer Sciences Corporation)
David Card	(Computer Sciences Corporation)

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 582.1
NASA/GSFC
Greenbelt, Md. 20771

ABSTRACT

This document is a glossary of terms used in the Software Engineering Laboratory (SEL). The terms are defined within the context of the software development environment for flight dynamics at Goddard Space Flight Center. The intent of this document is to provide a concise reference for clarifying and understanding the language employed in SEL documents and data collection forms.

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION

The glossary of Software Engineering Laboratory (SEL) terms presents a comprehensive collection of frequently used software engineering terms and expressions. Its objectives are to

- Provide a reference for clarifying the language of SEL documents and data collection forms
- Establish standard definitions for use by SEL personnel
- Explain basic software engineering concepts

The defintions provided for the terms in this document are the local (SEL) usages and have been compiled from many sources: SEL personnel, software engineering literature, and publications of computer software terminology.

SECTION 2 - SOFTWARE ENGINEERING TERMS

acceptance testing	Independent testing to verify acceptance criteria for program certification. The software pass/fail criteria are predetermined. Failure to meet all criteria causes rejection of the software product.
adaptability	A measure of the ease with which a program can be altered to fit differing user and system constraints.
adjusted lines of code	All new code plus 20 percent of the reused code, minus 50 percent estimated as the amount of comment lines, minus 10 percent estimated as the amount of nonexecutable statements. This measure is an estimate of the number of executable lines of code developed.
algorithm	A prescribed set of well-defined rules or processes for the solution of a problem.
analyzer	Computer software used as a tool that is applied to a program to provide analytical information; it breaks the program into identifiable segments and reports statistical information. This information can include execution frequency statistics, program path analysis, and/or source code syntax analysis.
archive	Process involving the transfer of data or information from one source or volume to another to provide a backup or alternate copy of the information for future use.
argument	Variable or expression passed to an operation or function as input or output.
array	An ordered group or collection of variables, terms, or expressions. An array is usually dimensioned (or indexed).
assemble	As done by an assembler.

assignment statement	An expression or instruction used to assign values to specified variables or symbols. Includes all statements that change the value of a variable as their main purpose (for example, read statements). However, the assignment of the iteration counter in a DO statement is not included.
attribute list	A compiler-generated list of the identifiers used by a program. It describes the characteristics of those identifiers and shows the source statements where they are defined (or used) and the (relative) storage locations of variables.
baseline diagram	A hierarchical graph of a software design listing all components in the system. A connection from a higher component to a lower one indicates that the higher component calls the lower one.
batch	Mode of operation of a computer in which the entire job is read into the machine before processing begins and in which there is no provision for interaction with the submitter during execution of the job.
block diagram	A diagram of a system or computer in which the principal parts are represented by geometrical figures that show both the basic functions and the functional relationships among the parts.
bug	An error in the design or implementation of a program. One or more software bugs exist in a system if a software change is required to meet specified or implied system performance requirements.
build	A functional subset of a more complex software development product. The "builds" approach to software development consists of developing a series of increasingly complete functional systems.

business and financial applications	Software or software system components related to some accounting, finance, or business data maintenance and reporting.
calibration error	An error in the gauge or tolerance of specifications.
certification test	A formal demonstration to the customer showing that requirements have been met.
change	A modification to requirements, design, code, or documentation made to correct an error, improve system performance, add capability, improve appearance, or implement a requirements change.
clerical error	An error made in the process of copying an item from one format to another or from one medium to another, which involves no interpretation or semantic translation.
code	A symbolic representation of a function composed of computer program statements.
code and unit test	Life cycle phase in which code is developed or modified to meet design specifications. Each module (or unit) is integrated into the system and tested to ensure that the newly added capabilities function correctly.
code reading	Inspection of the source code by persons other than the creator of the code in an attempt to detect errors.
coding	The generation of a symbolic representation of a function that can be executed by a computer.
command/control	A class of software including programs used either to generate vehicle commands or to transmit these commands from the control center.
compile	To translate a computer program expressed in a problem-oriented human readable language into a computer-oriented, machine executable language. This includes the function of an assembler. However, some compilers produce assembler rather than machine code.

complexity	A measure of the difficulty of implementing or understanding a component, independent of the implementor's experience; for example, the degree of interactions and number of dependencies among elements of a computer program.
component	A named piece of a system; for example, a separately compilable function, a functional subsystem, or a shared section of data such as a COMMON block.
computer architecture	The relationships between the parts of a computer system; the structural and functional definition of a computer as viewed in terms of its machine instruction set and input/output capabilities.
confidence level	The probability that a given statement is correct; 100 percent means that the statement is invariably true.
configuration control	A methodology for controlling the contents of a software system; a way of monitoring the status of system components, preserving the integrity of released and developing versions of a software system, and controlling the effects of changes throughout the system.
configuration management	All activities related to controlling the contents of a software system: monitoring the status of system components, preserving the integrity of released and developing versions of a system, and controlling the effects of changes throughout the system.
control statement	A statement that potentially alters the sequence of executed instructions (for example, GOTO, IF, RETURN, DO).
control structure	A recurrent pattern of control statements (for example, sequence, iteration, selection).
convention	An agreed-upon method, notation, or form of presentation.
correction	A change made to correct an error.

cosmetic change	A change in the source program made to improve clarity that has little effect on the performance of the program; for example, comment correction, movement of code that does not alter the implemented algorithm, or changing the name of a local variable.
cost estimation	Prediction of the amount of labor necessary to complete a task, the amount and potential costs of computer time required, etc., before and during a project's life cycle.
cost effective	A term used to describe a process deemed to perform a task correctly and completely with a minimum waste of resources.
costing technique	A method for determining the cost of developing a system or any particular part of a system.
criticality	A measure of the degree of dependence of the whole on a part of a system.
data	A series or collection of measurements.
data base	A set of data files that are logically related. An organized system of storing data.
data collection	The methods (that is, forms, procedures, personnel) for collecting measurements.
data definition language	A special-purpose language used to define data items in a data base and to create a data dictionary.
data dictionary	A file that describes the format of fields, values, and records in a data base.
data processing	A class of software whose primary function is the movement, formatting, and storage of data.
data set	A physical data storage location, usually magnetic tape or disk.
data structure	The logical relationship among the units of data in a data base.
data type	A set of attributes used to define a data item.

data validation	The process of verifying the completeness and accuracy of data.
data base management system (DBMS)	A software system for managing a data base, usually consisting of a data definition language and a data access language.
debugging	The process of locating and correcting software errors.
design	A description of what a system must do, its components, the interfaces among those components, and the system's interfaces with the external environment.
design language	A symbolic representation of a design, usually input to an analyzer program to detect errors and ambiguities.
design phase	The life cycle phase in which the structure of a system is planned and recorded.
design phase, preliminary	The specification of major functional subsystems, input/output interfaces, processing modes, and implementation strategy. The software system architecture is defined, based on the requirements given in the functional specification and requirements document, and translated into software requirements in the requirements analysis summary report.
design phase, detailed	The extension of the system architecture defined in the preliminary design phase to the subroutine level. The preliminary design is elaborated by successive refinement techniques to produce a "code-to" specification for the system.
design reading	Inspection of the design by persons other than the creator of the design.
design review	A formal meeting between customer and developer to determine that a proposed software configuration will satisfy performance specifications.
design specification	A document containing the approved design requirements for a program.

design verification	The formal examination or inspection of a software specification for the purpose of finding design errors and ambiguities.
development phase	The development and recording of code and inline comments based on the design. Includes the modification of code caused by design changes or errors found in testing. (See code and unit test.)
developed lines of code	The total number of new lines of source code plus 20 percent of reused code.
discrepancy	The difference between the intention of a specification and its actual implementation.
documentation	Written material, other than source code statements, that describes a system or any of its components.
driver	A software component developed specifically to call other components; used in an informal testing technique during the implementation phase.
dynamic allocation	The allocation of memory required by an operating program during its execution phase rather than prior to execution.
effectiveness	The degree to which a system can successfully meet an operational demand within a given time when operated under specified conditions.
efficiency	The ratio of useful work performed to the total energy expended. Code is efficient to the extent that it fulfills its purpose without wasting resources.
effort	The amount of resources, including manpower and computer time, necessary to complete a particular project; total energy expended.
element	A basic segment of a named piece of a system (component).
end date	The date that a phase is scheduled to be completed.

embedded system	A dedicated computer system that is physically incorporated into a larger system whose primary function is not data processing, for example, an electromechanical system.
environment	The combination of all external or extrinsic conditions that affect the operation of an entity. The combination of hardware and software used to maintain and execute the software, including the computer on which the software executes, the operating system for that computer, support libraries, text editors, compiler, etc.
error	An internal condition that prevents a software system from successfully performing its intended function. (See calibration error, clerical error.)
error analysis	The examination of errors with the purpose of tracing them to their sources and determining their effects.
error recovery	The ability of a system to resume processing rather than abort after an error.
estimation parameter	Any estimator or contributing factor to the process of estimation.
executable statement	Statement that changes the value of data or the state of a program.
execution	Performance by a computer of the instructions in a program.
execution time	The actual processor time used in executing a program.
external reference	A call to a function or subroutine in the source code that is outside the calling program body.
failure rate	The number of failures divided by the central processing unit (CPU) time for the interval. (See error rate.)
failure, software	An unacceptable result produced during the operation of the computer program. Occurs when a fault is evoked by some input data. (See error.)

fault	A specific manifestation of an error. A fault is evidenced when entry of some input data results in the program failing to correctly perform a required function.
file	A collection of data treated as a unit.
flight dynamics software	Applications to support attitude determination and control, maneuver planning, orbit adjustment, and general mission analysis.
flow chart	A graphical representation of an algorithm in which symbols are used to represent operations, data, data flow, equipment, etc.
form	Questionnaire used to record information about the software development process and/or software product.
- change report	Records software changes and error data.
- component status	Records time expended for activities.
- component summary	Records the status of system components.
- data base problem report	Used to identify and initiate action on data base problems.
- maintenance change report	Records software changes and error data.
- project summary	Used to classify the project and measure development progress.
- resource summary	Records expended resources.
- run analysis	Used to monitor activities for which the computer is used.
formal specification	A specification technique based on a strict set of rules for describing the specification and usually involving the use of an unambiguously defined notation (for example, mathematical functions or formal program design language (PDL)).
formal testing	Testing performed in accordance with customer-approved test plans. Verifies that the software system is operating according to the requirements of the development specifications.

format statement	A source language statement providing the necessary type and location information for read/write variables.
function	A mathematical subprogram used to specify an input set, an output set, and the relationship between the two.
functional specification	A specification of a software component as a set of functions defining the output for any input. Emphasizes what a program is to do rather than how to do it.
Halstead measures	Measures developed by M. Halstead in his theory of "software science," based on basic elements of programming languages: operator, operand, length, volume, and language level.
hardest first	The development approach of designing (or implementing) the most difficult aspects of a system first.
hardware	The physical and electronic components of a computer system including input/output devices, CPU, memory, etc.
hardware reliability	A measure of the probability of a hardware system operating without failure, usually measured as MTTF (mean time to failure).
hierarchy	A ranked series of elements, such as tasks, programs, people, functions, etc.
high-level language	A programming language that does not reflect the structure of any one given computer or that of any given class of computers.
HIPO (hierarchical input process output)	A software design technique that defines each component in terms of a transformation from an input data set to an output data set, usually represented in graphic form.
historical	Of or pertaining to data archives on past experience with particular projects.
identifier	A symbol whose purpose is to identify, indicate, name, or locate a data structure or procedure entry point.

impact	The magnitude of effort or effect associated with a particular task or change in requirements, software, etc.
implementation	Development phase involving code and unit testing. (See code and unit test.)
informal testing	Testing involving no formal, written test plan.
input/output (I/O)	Usually refers to data or hardware processes involving the transfer of information to or from computer main memory.
instruction	(See executable statement.)
integration	The combination of subunits into an overall unit or system by means of interfacing.
integration test	A test of several modules to check that the interfaces are implemented correctly.
interactive	A mode of computer operation in which each line of input is immediately processed; allows communication with the program during its execution.
interface	The set of data and control information passed between two or more programs or segments of programs and the assumptions made by each program about how the others operate.
interface testing	Validation that a module or set of modules operates within agreed interface specifications to ensure proper data and logical communications.
interpret	To translate and execute one step (statement) at a time; to execute high-level language programs by translating each statement to a corresponding sequence of machine operations before proceeding to the next statement.
interrupt	Any stopping of a process in such a way that it can be resumed.
iteration	Repetition of a sequence of instructions until a specified set of conditions is satisfied.

iterative enhancement	The design (or implementation) of successive versions, each producing a usable subset of the final product, until the entire system is fully developed.
IV&V	A software methodology employing independent verification and validation techniques.
job	A unit of computer work consisting of one or more steps such as compilation, assembly, or utility runs.
job control language (JCL)	A program language controlling the use of computer system resources.
level of effort	Effort expended as needed and available.
librarian	Programming support personnel whose responsibilities include processing source statements but not writing them (for example, maintaining libraries, updating code, and producing tape backups).
life cycle	Sequence of phases during which the software product is developed from concept through testing and completion. (See individual phases: pretask planning, requirements analysis, preliminary design, detailed design, code and unit testing, system integration and testing, acceptance testing, maintenance and operation.)
lines of code (LOC)	Eighty character card images of source code (programming language statements)
- adjusted	An estimate of the number of executable lines of code. Includes all new code plus 20 percent of the reused code, minus 50 percent estimated as the amount of comment lines, minus 10 percent estimated as the amount of nonexecutable statements.
- delivered	Total number of lines of source code generated as a deliverable item for a project. Includes executable, nonexecutable, and comment statements and all statements newly coded as well as statements taken from existing programs and library routines.

- developed	Total number of new lines of source code plus 20 percent of reused code.
- executable	Code that changes the value or state of a program or data.
- modified	Changed, existing code.
- new	Total number of lines of source code written by programmers for a given task. Does not include any code that was taken from previously existing programs, but does include comments, executable, and nonexecutable statements.
- old	Total number of lines of source code taken from previously existing programs. Sometimes refers to reused unchanged.
- reused	Same as old lines of code.
load module	An executable program produced by translating and linking source code.
machine language	A system of numeric operation codes, values, and addresses, a sequence of which can be directly executed by a computer.
macro	A single instruction in a source language that represents a defined sequence of source instructions in the same language. A macro is replaced by the sequence it represents before program translation.
main program	A program unit containing at least one executable statement and having a starting address for program execution; normally, the set of instructions that determines the basic sequence of control.
maintenance	The process of modifying existing operational software to correct errors or enhance capabilities while leaving its primary function intact.
management, software	All the technical and management activities, decisions, and controls directly required to purchase, develop, or maintain software throughout the life cycle and maintenance phases.

management, technical	Planning, organization, motivation (direction), and control of a technical project and technical personnel.
manpower	The level or amount of total human effort required or used for a project.
measure	A count or numerical rating of the occurrence of some property. Examples include lines of code, number of computer runs, person-hours expended, and degree of use of top-down design methodology.
methodology	A prescribed set of principles for the development process. These principles may pertain to requirements, design, code, testing, or management. Examples include structured analysis, top-down design, information hiding, structured programming, formal test plans, and configuration management.
metric	(See measure.)
microcomputer	A class of computer having all major central processor functions contained on a single integrated circuit (MPU). Typically implemented as the MPU plus a small number of supporting integrated circuits and characterized by a word size not exceeding 16 bits.
microprocessor	A single integrated circuit (MPU) that performs the functions of a central processing unit (CPU).
mission date	The date that the system must be operational, usually 2 months before launch.
model	Equation relating two or more quantitative factors. A resource utilization model may provide an estimate of the cost of a project; a reliability model may indicate when sufficient testing has been done.
modification	The process of altering a program and its specification to perform either a new task or a different but similar task.

modified code	(See lines of code.)
module	A named subprogram unit that is independently compilable.
module test	The test of a single module.
new lines of code	(See lines of code.)
object module	A computer program expressed in machine language, usually the result of translating a source program by an assembler or compiler.
online processing	Interactive processing, between humans and the computer.
operand	(See Halstead measures.)
operator	(See Halstead measures.)
operating system	A system of routines and services that monitors, controls, allocates, deallocates, and manages system resources and the execution of application programs and other system routines.
operation	A function that transforms data objects from input domain(s) into data objects in the operation's output domain(s).
optimization	A change in the source code to improve program performance, for example, to make it run faster or use less space. Optimization changes are not error corrections; however, if a change is made to use less space in order to conform to a specified space constraint, the term "error" applies.
overlay	A hierarchical structure of program components that allows the program to be executed while only part of it is in memory at any given time.
parameter	A variable or measure that can take on more than one value, but only one at a time.
parse	To decompose a sequence of symbols unit (block, line, phrase, word) into a set of elementary subunits (lines, words, commands, characters).
phase	(See life cycle.)

precompiler	A computer program used to add special-purpose capabilities to a language system. A precompiler translates special features implemented as macros into regular instruction sequences in a programming language.
preliminary design	(See design phase.)
pretask planning	Planning efforts leading up to requirements analysis; development of software development plans and estimates.
preventive maintenance	Maintenance specifically intended to prevent faults from occurring.
procedure	A sequence of steps that accomplishes some task.
procedural specification	A specification of a software component in an algorithmic manner, stating how the program is to work.
process design language	(See program design language.)
productivity	Generally accepted as the quantity of code produced (lines of code per man-month) or the rate of production of computer software measured in the quantity of code and documentation produced.
program	A sequence of instructions that directs the computer to perform a task.
program complexity	A function of the number of execution paths in the program and the difficulty of determining the path for an arbitrary set of input data. (See complexity.)
program design language (PDL)	A language, often called pseudocode, used in the design and coding phases of a project, that contains a fixed set of control statements and a formal or informal way of defining and operating on data structures.
program listing	The sequence of instructions making up a computer program, usually in the form of a printout.

program validation	All techniques used to ensure correct programs, including system, and subsystem, and system integration testing.
programming language	A formal language composed of statements and instructions that has a formal syntax and lexical rules and that can be used in composing computer programs that require translation to be machine executable.
project	A software development effort with set goals and defined objectives that uses the technical and managerial capabilities of personnel, has a life cycle with fixed endpoints, and produces a specified product.
proof technique	A method for formally demonstrating that a piece of software performs according to its specifications. Proof techniques usually use some form of mathematical notation to describe the result of executing a program.
prototype	A system developed with the intention of serving as a pattern for a future development effort.
quality	The degree to which software conforms to certain desirable characteristics. These may include, but are not limited to, correctness, reliability, usability, validity, efficiency, flexibility, and maintainability.
quality assurance	A planned and systematic procedure for ensuring that the product conforms to established technical requirements and quality standards.
read	The reading by peers of code and design materials to look for errors, invent tests, and so on.
real-time	A program that receives input from a process or activity and reacts in time to affect that process or activity.
reliability	The probability that software will function without failure.
requirement	A system specification written by the user to define a system to a developer. The developer uses this specification in designing, implementing, and testing the system.

requirements analysis	An analysis of the contents of the functional specification and requirements document from a software system viewpoint, to recast the requirements in terms suitable for software design. The completeness and feasibility of the requirements are assessed; missing or to-be-determined requirements are identified; all external interfaces are specified; and the initial determination and allocation of resources are made.
requirements testing	The execution of a software product under controlled conditions to demonstrate that all stated or implied requirements and performance criteria have been met.
resource	Any person, equipment, or facility that may be allocated to the accomplishment of a task.
resource estimation model	A model that attempts to relate measures of manpower and/or computer time to measures of the software problem, product, process, and environment. May range from simple, single variable equations to complex interactive software packages.
reused code	(See lines of code.)
review	A formal meeting of several individuals for the purpose of examining design, requirements, code (management review).
routine	A subprogram or module.
scheduling	The allocation of time, and resources necessary to complete a given task or project.
segment	A contiguous piece of code that is unnamed and, hence, cannot be referred to as a single entity in a program statement. Could be one or several lines of a routine, subroutine, part of a data area, or an arbitrary contiguous section of memory.

Software Engineering Laboratory (SEL)	An organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members: NASA/GSFC (Systems Development and Analysis Branch), The University of Maryland (Computer Sciences Department), and Computer Sciences Corporation (Flight Systems Operation). The goals of the SEL are to understand the software development process in the GSFC environment; to measure the effect of various methodologies, tools, and models on this process; and to identify and then to apply successful development practices.
shared items	Data and programs accessible by several components, such as COMMON blocks, external files, and library subroutines.
simulated constructs	Statements used to simulate structured control structures when the language to be used does not contain these structures.
software	Computer program code and its associated data, documentation, and operational procedures.
software class	The type of software according to content and purpose: scientific, data processing, or control.
software development life cycle	(See life cycle.)
software engineering	The scientific approach to software development employing proven cost-effective methodologies, tools, and techniques.
software reliability	(See reliability.)
software testing	The process of exercising software in an attempt to detect errors that exist in the code. (See formal testing.)

source statements	All statements input to a compiler. Includes executable statements (assignment, IF, and GO TO); nonexecutable statements (DIMENSION, REAL, and END); and comments.
specification	A description of the input, output, and essential function(s) to be performed by a component of the system. Produced by the organization that is to develop the system; that is, it can be thought of as the contractor's interpretation of the requirements.
- imprecise	The input, output, and function of the component are loosely defined. Much of what is required is assumed rather than specified. The specification relies heavily on programmer experience and verbal communication to get an unambiguous interpretation and a full understanding of what is needed.
- precise	The input, output, and function of the component are well defined. There are underlying assumptions not specified, but it is assumed that any programmer working on the project, with experience on a similar project, will understand these assumptions. It is possible to arrive at an ambiguous interpretation or misunderstanding of the specifications if the reader does not have enough experience with the problem or does not obtain further verbal communication.
- very precise	A completely defined description of the input, output, and function of a component. The implementer of a very precise specification need make few, if any, assumptions. It is almost impossible to arrive at an ambiguous interpretation or misunderstanding of the specifications.
specification-driven	Uses the specifications of the program to determine test data; for example, generating test data by examining the input/output requirements and specifications).

staff-units	A concept used to estimate or measure human energy expended on a particular project. Based on the length of a working day, 6 or 8 hours productive time or calendar time (for example, staff-months, staff-hours).
standard	Any specification that refers to the method of development of the source program itself, and not to the problem to be implemented (for example, using structured code, at most 100-line subroutines, or all names prefixed with subsystem name).
string processing	Includes components that perform operations on lists of characters. Normally assumed to include functions of compilers, hash code string hookup, and array comparisons.
structure-driven	Uses the structure of the program to determine test data; (for example, generating data to ensure that each branch of a program is executed at least once).
structured code	Code that uses only the structured constructs: DO WHILE (iteration), IF-THEN-ELSE (selection), and BEGIN-END (sequence).
structured design	The use of a modular, hierarchical design consistent with structured coding practices. A set of techniques for reducing the complexity of large new programs by dividing them into independent modules.
structured programming	Programming with a limited set of constructs; programming with structured code.
stub	A "dummy" software element used in place of an expected functional element until the expected element becomes available.
subprogram	A module, separately compilable but not independently executable; a collection of program elements that provides a function or relatively independent functions with respect to the whole program.

subroutine	A subprogram that does not return a value associated with its name when invoked.
subsystem	A collection of subprograms that provides a major function and is independent of any other subsystem.
support software	All programs used in the development and maintenance of the delivered operational programs.
systems software	Any package designed to affect, modify, extend, or change the normal available processing procedure of the operating system. Could include such components as error tracing or extended input/output such as DAIO.
system	A set or arrangement of software or hardware related or connected to form a unity capable of achieving the goals specified in its design.
system description	A document illustrating system baselines, data flows, and processing descriptions.
system integration	The process of combining system components to produce the total system.
system size	The total number of machine words needed for all instructions generated on the project plus space for data, library routines, and other codes; the total size of the system without using any overlay structure.
system test	The process of trying to find discrepancies between the system and its original objectives.
table handler	Components that are specifically designed to generate or interpret information stored in a table format, such as the Generalized Telemetry Processor.
task	A set of defined objectives. Multiple tasks are initiated to complete a project. (See project.)
TBD	To be determined.
technical management	(See management.)

telemetry	Data transmitted at regular intervals from sensors.
test	A procedure designed to verify some aspect of the performance of a software system.
test plan	A description of test conditions that includes inputs, expected outputs, parameter values, etc.
test plan document	A management document that describes how and when specified test objectives will be met for the formal test plan.
testing	Part of the software development process in which a software system is subjected to specific conditions to show that it meets the intended design.
- functional	The execution of independent tests designed to demonstrate a specific functional capability of a program or software system.
- unit	Test of a set of program statements treated logically as a whole.
timesharing	A mode of operation that provides for the interleaving of two or more independent processes on one functional unit.
tool	A software aid used during the automated development process to facilitate the work of development team members. Examples are requirements language processors, precompilers, code auditors, and test generators.
top-down development	The design (or implementation) of the system, starting with a single component, one level at a time, by expanding each component reference as an algorithm possibly calling other new components.
top-down testing	Testing of modules that were produced in top-down order.
tree chart	An acyclic connected graph, often representing a hierarchy in which the edges are directed to denote a subordinating relationship between the joined nodes.

uncertainty	The probability of error, or the probable magnitude of error.
unit	A set of computer program statements treated logically as a whole; usually a module or subroutine. (See component.)
unit test	Independent test of a unit. (See implementation and module test.)
user	The individual at the man/machine interface who is applying the software to the solution of a problem.
user-defined	An entity determined by the user as input during program execution.
user's guide	A document designed to assist the user in operating the software product.
utility	Any component that is generated to satisfy some general support function required by other applications software.
validation	The process of determining whether executing the system in a user environment causes any operational difficulties. The process includes ensuring that specific program functions meet their requirements and specifications.
verification	The process of determining whether the results of executing the software product in a test environment agree with its specifications.
walk-through	A formal meeting for the review of source code and/or design by project members for the purpose of error detection, not correction; a technical rather than management review.
work-around	The method used to counteract the effects of an error in a program when the cause of the error and, consequently, the location of the statements containing the error is not known or is inaccessible (for example, a compiler error).

work unit	A quantity defined to enable an estimator to break down project requirements, and subsequently cost, into quantifiable deliverable items. Some common work units include the number of requirements, programs, subsystems, modules, pages of documentation, lines of code, and experience of developers.

SECTION 3 - ACRONYMS

ALC	Assembly Language Code
ATR	Assistant Technical Representative
BMDP	Biomedical Programs, P Series
CAREM	Cost and Resource Estimating Models
CAT	Configuration Analysis Tool
COCOMO	Constructive Cost Model
CSC	Computer Sciences Corporation
DARES	Data Base Retrieval System
DBA	Data Base Administrator
DBAM	Data Base Maintenance Software
GESS	Graphic Executive Support System
GSFC	Goddard Space Flight Center
HIPO	Hierarchical Input Processing Output
IV&V	Independent Verification and Validation
MPP	Modern Programming Practices
MTTF	Mean Time to Failure
PANVALET	Computer Program Analysis and Security System
PDL	Program/Process Design Language
PRICES	Programmed Review of Information for Costing and Evaluation Software Model
SAP	FORTRAN Static Source Code Analyzer Program
SEL	Software Engineering Laboratory
SFORT	Structured FORTRAN Preprocessor
SLIM	Software Life-Cycle Management Estimating Model
STL	Systems Technology Laboratory
TBD	To Be Determined
TSO	IBM Timesharing Option
UM	University of Maryland

BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-Originated Documents

SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu and D. S. Wilson, September 1977

SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

†SEL-78-002, FORTRAN Static Source Code Analyzer (SAP) User's Guide, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

SEL-78-102, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1), W. J. Decker and W. A. Taylor, September 1982

SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

†This document superseded by revised document.

SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson and B. Chu, September 1978

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-001, Functional Requirements/Specifications for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, A. L. Green, and C. E. Goorevich, February 1980

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-004, System Description and User's Guide for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, W. J. Decker, J. G. Garrahan, et al., October 1980

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

†SEL-81-001, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-002, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, G. Page, and F. E. McGarry, September 1981

SEL-81-003, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, D. N. Card, D. C. Wyckoff, and G. Page, September 1981

†SEL-81-004, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., September 1981

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

†SEL-81-005, Standard Approach to Software Development, V. E. Church, F. E. McGarry, G. Page, et al., September 1981

SEL-81-105, Recommended Approach to Software Development, S. Eslinger, F. E. McGarry, and G. Page, May 1982

SEL-81-006, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, December 1981

†SEL-81-007, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, E. J. Smith, A. L. Green, et al., February 1981

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

†This document superseded by revised document.

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

SEL-81-010, Performance and Evaluation of an Independent Software Verification and Integration Process, G. Page and F. E. McGarry, May 1981

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve As a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-82-001, Evaluation of Management Measures of Software Development, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-002, FORTRAN Static Source Code Analyzer Program (SAP) System Description, W. A. Taylor and W. J. Decker, August 1982

SEL-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo, September 1982

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

SEL-82-005, Glossary of Software Engineering Laboratory Terms, M. G. Rohleder, December 1982

SEL-82-006, Annotated Bibliography of Software Engineering Laboratory (SEL) Literature, D. N. Card, November 1982

SEL-Related Literature

††Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Banks, F. K., "Configuration Analysis Tool (CAT) Design," Computer Sciences Corporation, Technical Memorandum, March 1980

††Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation," University of Maryland, Technical Memorandum, October 1979

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: Computer Societies Press, 1980 (also designated SEL-80-008)

††Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

††Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and B. T. Perricone, Software Errors and Complexity: An Empirical Investigation, University of Maryland, Technical Report TR-1195, August 1982

††Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

Basili, V. R., R. W. Selby, and T. Phillips, Metric Analysis and Data Validation Across FORTRAN Projects, University of Maryland, Technical Report, November 1982

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity and Cost, October 1979

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

††Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

††Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: Computer Societies Press, 1978

††Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

Card, D. N., and M. G. Rohleder, "Report of Data Expansion Efforts," Computer Sciences Corporation, Technical Memorandum, September 1982

††Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

Freburger, K., "A Model of the Software Life Cycle" (paper prepared for the University of Maryland, December 1978)

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

Hislop, G., "Some Tests of Halstead Measures" (paper prepared for the University of Maryland, December 1978)

Lange, S. F., "A Child's Garden of Complexity Measures" (paper prepared for the University of Maryland, December 1978)

Miller, A. M., "A Survey of Several Reliability Models" (paper prepared for the University of Maryland, December 1978)

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (proceedings), March 1980

Page, G., "Software Engineering Course Evaluation," Computer Sciences Corporation, Technical Memorandum, December 1977

Parr, F., and D. Weiss, "Concepts Used in the Change Report Form," NASA, Goddard Space Flight Center, Technical Memorandum, May 1978

Reiter, R. W., "The Nature, Organization, Measurement, and Management of Software Complexity" (paper prepared for the University of Maryland, December 1976)

Scheffer, P. A., and C. E. Velez, "GSFC NAVPAK Design Higher Order Languages Study: Addendum," Martin Marietta Corporation, Technical Memorandum, September 1977

Turner, C., and G. Caron, A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, NASA/SEL Data Compendium, Data and Analysis Center for Software, Special Publication, April 1981

Weiss, D. M., "Error and Change Analysis," Naval Research Laboratory, Technical Memorandum, December 1977

Williamson, I. M., "Resource Model Testing and Information," Naval Research Laboratory, Technical Memorandum, July 1979

††Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: Computer Societies Press, 1979

Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings), November 1982

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.